UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report: Aitech Software Engineering Ltd., AI-ADA/020, Version 1.0, DEC MicroVAX II to a Motorola MVME 133 board (MC68020), 880520W1.09061

5. TYPE OF REPORT & PERIOD COVERED

30 May 1988 to 30 May 1988

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Wright-Patterson AFB
Dayton, OH

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

Wright-Patterson AFB
Dayton, OH

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)

Wright-Patterson AFB
Dayton, OH

15. SECURITY CLASS (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Aitech Software Engineering Ltd., AI-ADA/020, Version 1.0, DEC MicroVAX II under MicroVMS, Version 4.5 (Host) to Motorola MVME 133 board (MC68020) (bare machine), ACVC 1.9, Wright-Patterson AFB.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: AVF-VSR-159-0788
87-12-14-AIT

Ada COMPILER
VALIDATION SUMMARY REPORT:

Certificate Number: 880520W1.09061
Aitech Software Engineering Ltd.
AI-ADA/020, Version 1.0
DEC MicroVAX II to a Motorola MVME 133 board (MC68020)

Completion of On-Site Testing:
30 May 1988

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington D.C. 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: AI-ADA/020, Version 1.0

Certificate Number: 880520W1.09061

Host:	Target:
DEC MicroVAX II under MicroVMS, Version 4.5	Motorola MVME 133 board (MC68020) (bare)

Testing Completed 30 May 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Steven P. Wilson

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

William S. Ritchie

Ada Joint Program Office
William S. Ritchie
Acting Director
Department of Defense
Washington, DC 20301

Ada Compiler Validation Summary Report:

Compiler Name: AI-ADA/020, Version 1.0

Certificate Number: 880520W1.09061

Host:
DEC MicroVAX II
under MicroVMS, Version 4.5

Target:
Motorola MVME 133 board (MC68020)
(bare)

Testing Completed 30 May 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Steven P. Wilson

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by Softech, Inc., under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 30 May 1988 at Herzelia, Israel.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. *Ada Compiler Validation Procedures and Guidelines*, Ada Joint Program Office, 1 January 1987.

3. *Ada Compiler Validation Capability Implementers' Guide*, SofTech, Inc., December 1986.

4. *Ada Compiler Validation Capability User's Guide*, December 1986.

1.4 DEFINITION OF TERMS

ACVC
The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary
An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard
ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant
The agency requesting validation.

AVF
The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the *Ada Compiler Validation Procedures and Guidelines*.

AVO The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host The computer on which the compiler resides.

Inapplicable test An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test An ACVC test for which a compiler generates the expected result.

Target The computer for which a compiler generates code.

Test A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: AI-ADA/020, Version 1.0

ACVC Version: 1.9

Certificate Number:	880520W1.09061
Host Computer:	
Machine:	DEC MicroVAX II
Operating System:	MicroVMS, Version 4.5
Memory Size:	9 megabytes
Target Computer:	
Machine:	Motorola MVME 133 board (MC68020)
Operating System:	(bare)
Memory Size:	1 megabyte
Communications:	RS-232

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER, SHORT_FLOAT, and LONG_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

  Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is gradual. (See tests C45524A..Z.)}

- Rounding.

The method used for rounding to integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

- Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR only for a two-dimensional array when the big dimension is the second one. Otherwise, no exception is raised. (See test C36003A.)

CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST}. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

Not all choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are not supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are not supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures and functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO can be instantiated with unconstrained array types and unconstrained record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

- Generics.

  Generic declarations and bodies can be compiled in separate compilations only when the instantiation occurs after the body. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

  Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 had been withdrawn because of test errors. The AVF determined that 363 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 159 executable tests that use floating-point precision exceeding that supported by the implementation and 172 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for eight tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	107	1048	1501	17	13	46	2732
Inapplicable	3	3	352	0	5	0	363
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	187	514	572	248	166	98	140	326	135	36	232	3	75	2732
Inapplicable	17	58	102	0	0	0	3	1	2	0	2	0	178	363
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C
C35904A	C35904B	C35A03E	C35A03R	C37213H
C37213J	C37215C	C37215G	C37215E	C37215H
C38102C	C41402A	C45332A	C45614C	A74106C
C85018B	C87B04B	CC1311B	BC3105A	AD1A01A
CE2401H	CE3208A			

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 363 tests were inapplicable for the reasons indicated:

- C24113I..N (6 tests) are inapplicable because they contain real literals whose lengths exceed this implementation's maximum line length.

- A39005C and C87B62B use a STORAGE_SIZE clause for an access type, which is not supported by this implementation.

- A39005D and C87B62D use a STORAGE_SIZE clause for a task type, which is not supported by this implementation.

- C45231D and B86001D require a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This implementation does not support any such types.

- C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this implementation.

- C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this implementation.

- C4A013B uses a static value that is outside the range of the most accurate floating-point base type. The declaration was rejected at compile time.

- C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.

- CA2009C and BC3205D compile generic package specifications and bodies in separate compilations. This implementation requires that generic package specifications and bodies be in a single compilation.

- CA2009F and BC3204C compile generic subprogram declarations and bodies in separate compilations. This implementation requires that generic subprogram declarations and bodies be in a single compilation.

- The following 178 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102C CE2102G..H(2) CE2102K CE2104A..D(4)
CE2105A..B(2) CE2106A..B(2) CE2107A..I(9) CE2108A..D(4)
CE2109A..C(3) CE2110A..C(3) CE2111A..E(5) CE2111G..H(2)
CE2115A..B(2) CE2201A..C(3) EE2201D..E(2) CE2201F..G(2)
CE2204A..B(2) CE2208B CE2210A CE2401A..C(3)
EE2401D CE2401E..F(2) EE2401G CE2404A
CE2405B CE2406A CE2407A CE2408A
CE2409A CE2410A CE2411A AE3101A
CE3102B EE3102C CE3103A CE3104A
CE3107A CE3108A..B(2) CE3109A CE3110A
CE3111A..E(5) CE3112A..B(2) CE3114A..B(2) CE3115A
CE3203A CE3301A..C(3) CE3302A CE3305A
CE3402A..D(4) CE3403A..C(3) CE3403E..F(2) CE3404A..C(3)
CE3405A..D(4) CE3406A..D(4) CE3407A..C(3) CE3408A..C(3)
CE3409A CE3409C..F(4) CE3410A CE3410C..F(4)
CE3411A CE3412A CE3413A CE3413C
CE3602A..D(4) CE3603A CE3604A CE3605A..E(5)
CE3606A..B(2) CE3704A..B(2) CE3704D..F(3) CE3704M..O(3)
CE3706D CE3706F CE3804A..E(5) CE3804G
CE3804I CE3804K CE3804M CE3805A..B(2)
CE3806A CE3806D..E(2) CE3905A..C(3) CE3905L
CE3906A..C(3) CE3906E..F(2)

- The following 159 tests require a floating-point accuracy that exceeds the maximum of 18 digits supported by this implementation:

C24113O..Y (11 tests)
C35706O..Y (11 tests)
C35705O..Y (11 tests)
C35707O..Y (11 tests)
C35708O..Y (11 tests)
C35802O..Z (12 tests)
C45241O..Y (11 tests)
C45321O..Y (11 tests)
C45421O..Y (11 tests)
C45521O..Z (12 tests)
C45524O..Z (12 tests)
C45621O..Z (12 tests)
C45641O..Y (11 tests)
C46012O..Z (12 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required to the following eight Class B tests because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B33301A B55A01A B67001A B67001C B67001D
BC1109A BC1109C BC1109D

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the AI-ADA/020, Version 1.0, Ada compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and that the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the AI-ADA/020, Version 1.0, Ada compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a DEC MicroVAX II host operating under MicroVMS, Version 4.5, and a bare Motorola MVME 133 board (MC68020) target. The host and target computers were linked via RS-232.

A cartridge tape containing all of the tests, except for the withdrawn tests and the tests requiring unsupported floating-point precisions, was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled and linked on the DEC MicroVAX II, and all executable tests were run on the Motorola MVME 133 board (MC68020). Object files were linked on the host computer, and executable images were transferred to the target computer via RS-232. Results were transferred to the host computer from where they were transferred to a VAX-11/750 and printed.

The compiler was tested using command scripts provided by Aitech Software Engineering Ltd., and reviewed by the validation team. The compiler was tested using all default switch settings except for the following:

Switch	Effect
PROGRESS	shows compiler progress during compilation
LIST	produces a compilation listing

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Herzelia, Israel and was completed on 30 May 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

Aitech Software Engineering Ltd. has submitted the following Declaration of Conformance concerning the AI-ADA/020, Version 1.0, Ada compiler.

aitech

DECLARATION OF CONFORMANCE

Compiler Implementer: Aitech Software Engineering Ltd.
Ada Valdation Facility: ASD/SCEL, Wright-Patterson AFB
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name:	AI-ADA/020	Version: 1.0
Host Architecture:	ISA: MicroVAX II	OS&VER#MicroVMS, Version 4.5
Target Architecture:	ISA: Motorola MVME 133 board	OS&VER#bare

Implementer's Declaration

I, the undersigned, representing Aitech Software Engineering Ltd. have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler tested in this declaration. I declare that Aitech Software Engineering Ltd. is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

30-May-1988

Daniel Kord
Project Manager

Date

Owner's Declaration

I, the undersigned, representing Aitech Software Engineering Ltd. take full responsibilty for implementation and maintenance of Ada compiler listed above, and agree to public disclosure of the final Validation Summary Report. I declare that the Ada language compiler listed and its host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

30-May-1988

Daniel Kord
Project Manager

Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the AI-ADA/020, Version 1.0, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -32768 .. 32767;
     type SHORT_INTEGER is range -128 .. 127;
     type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_648;

     type FLOAT is digits 15 range -1.70141E+38 .. 1.70141E+38;
     type SHORT_FLOAT is digits 6
                     range -1.6777215E+31 .. 1.6777215E+31;
     type LONG_FLOAT is digits 18
                range -1.615850303565550364860552993479784444300154 2E+616
                  .. +1.6158503035655503648605529934797844443001542E+616;
     type DURATION is delta 2#1.0#E-14 range -131_072.0 .. 131_071.0;

     ...

end STANDARD;
```

F Appendix F of the Ada Reference Manual

F.0. Introduction

This appendix describes the implementation-dependent characteristics of the AI_ADA/020 Cross Compiler, as required in the Appendix F frame of the Ada Reference Manual (ANSI/MIL-STD-1815A).

F.1 Implementation-Dependent Pragmas

No implementation-dependent pragmas are defined for AI_ADA/020.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined for AI_ADA/020.

F.3 Package SYSTEM

The specification of the package SYSTEM:

```
package SYSTEM is

    type ADDRESS        is private;
    subtype PRIORITY    is INTEGER range 1..23;
    type NAME           is (M68020,M68000);
    SYSTEM_NAME:        constant NAME  := M68020;
    STORAGE_UNIT:       constant    := 16;
    MEMORY_SIZE:        constant    := 2048 * 1024;
    MIN_INT:            constant    := -2_147_483_647_1 - 1
    MAX_INT:            constant    := 2_147_483_647;
    MAX_DIGITS:         constant    := 18;
    MAX_MANTISSA:       constant    := 31;

    FINE_DELTA:         constant       := 2#1.0#E-31;
    TICK:               constant       := 0.000_001;

end SYSTEM;
```

F.4 Representation Clauses

Representation clauses are supported with the following limitations:

Representation clauses are not allowed for some kinds of derived types.

Length clauses with STORAGE_SIZE are not supported.

Length clauses for enumeration types are limited to sizes 8 or 16 bits.

Enumeration representation clauses are supported only within the range of type INTEGER.

Record representation clauses are not supported.

F.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler

F.6 Address Clauses

Not supported by the compiler

F.7 Unchecked Conversion

Not supported by the compiler.

F.8 Input-Output Packages

The implementation supports all requirements of the Ada language.

F.8.1 External Files and File Objects

External files are not supported.

F.8.2. Sequential and Direct Files

Sequential and direct files are not supported. When attempting to access a file, the appropriate exception is raised (see LRM: Chapter 14).

F.8.2.1. Specification of the Package Sequential IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

generic

     type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

     type FILE_TYPE is limited private;

     type FILE_MODE is (IN_FILE, OUT_FILE);

     File management

     procedure CREATE(FILE : in out FILE_TYPE;
                      MODE :        FILE_MODE := OUT_FILE;
                      NAME :        STRING    := "";
                      FORM : in     STRING    := "");

     procedure OPEN  (FILE : in out FILE_TYPE;
                      MODE : in     FILE_MODE;
                      NAME : in     STRING;
                      FORM : IN     STRING := "");

     procedure CLOSE (FILE : in out FILE_TYPE);

     procedure DELETE(FILE : in out FILE_TYPE);
```

F.8.2.1. (Continued)

```
     procedure RESET (FILE : in out FILE_TYPE;
                      MODE : in     FILE_MODE);

     procedure RESET (FILE : in out FILE_TYPE);

     function MODE   (FILE : in FILE_TYPE) return FILE_MODE;

     function NAME   (FILE : in FILE_TYPE) return STRING;

     function FORM   (FILE : in FILE_TYPE) return STRING;

     function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

--   input and output operations

     procedure READ  (FILE : in     FILE_TYPE;
                      ITEM :    out ELEMENT_TYPE);

     procedure WRITE (FILE : in FILE_TYPE;
                      ITEM : in ELEMENT_TYPE);

     function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;


--   exceptions

     STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
     MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
     NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
     USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
     DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
     END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
     DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

     type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end SEQUENTIAL_IO;
```

F.8.2.2 Specification of the Package Direct IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

generic

     type ELEMENT_TYPE is private;

package DIRECT_IO is

     type FILE_TYPE is limited private;

     type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);

     type COUNT is range 0..LONG_INTEGER'LAST;
     subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;
```

F.8.2.2 (Continued)

```
--    File management

      procedure CREATE(FILE : in out FILE_TYPE;
                       MODE : in     FILE_MODE   := INOUT_FILE;
                       NAME : in     STRING      := "";
                       FORM : in     STRING      := "");

      procedure OPEN  (FILE : in out FILE_TYPE;
                       MODE : in     FILE_MODE;
                       NAME : in     STRING;
                       FORM : in     STRING      := "" );

      procedure CLOSE (FILE : in out FILE_TYPE);

      procedure DELETE(FILE : in out FILE_TYPE);

      procedure RESET (FILE : in out FILE_TYPE;
                       MODE : in     FILE_MODE);

      procedure RESET (FILE : in out FILE_TYPE);

      function MODE   (FILE : in FILE_TYPE)   return FILE_MODE;

      function NAME   (FILE : in FILE_TYPE)   return STRING;

      function FORM   (FILE : in FILE_TYPE)   return STRING;

      function IS_OPEN(FILE : in FILE_TYPE)   return BOOLEAN;

--    input and output operations

      procedure READ  (FILE : in       FILE_TYPE;
                       ITEM :    out   ELEMENT_TYPE;
                       FROM : in       POSITIVE_COUNT);
      procedure READ  (FILE : in       FILE_TYPE;
                       ITEM :    out   ELEMENT_TYPE);
```

F.8.2.2 (Continued)

```
     procedure WRITE (FILE : in FILE_TYPE)
                      ITEM : in ELEMENT_TYPE;
                      TO   : in POSITIVE_COUNT);
     procedure READ  (FILE : in FILE_TYPE;
                      ITEM : in ELEMENT_TYPE);

     procedure SET_INDEX(FILE :  in FILE_TYPE;
                         TO   :  POSITIVE_COUNT

     function  INDEX(FILE : in FILE_TYPE) return POSITIVE_COUNT;

     function  SIZE (FILE : in FILE_TYPE) return COUNT;

     function  END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;

--   exceptions

     STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
     MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
     NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
     USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
     DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
     END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
     DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

     type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

F.8.3 Text Input-Output.

Only Standard input and Standard Output are supported.

F.8.3.1 Specification of the Package Text IO.

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

package TEXT_IO is

     type FILE_TYPE is limited private;

     type FILE_MODE is (IN_FILE, OUT_FILE);

     type      COUNT is range 0 .. LONG_INTERGER°LAST;
     subtype   POSITIVE_COUNT is COUNT range 1 .. COUNT°LAST;
     UNBOUNDED: constant COUNT:= 0; -- line and page length

     subtype FIELD       is INTEGER range 0 .. 35; -- max.  size     of   an
 integer output field
                                                   -- 2#....#
     subtype NUMBER_BASE      is INTEGER range 2 .. 16;
```

F.8.3.1 (Continued)

```
     type TYPE_SET is (LOWER_CASE, UPPER_CASE);

pragma PAGE;
     -- File Management

     procedure CREATE (FILE : in out FILE_TYPE;
                       MODE : in     FILE_MODE := OUT_FILE;
                       NAME : in     STRING    := "";
                       FORM : in     STRING    := ""
                      );

     procedure OPEN   (FILE : in out FILE_TYPE;
                       MODE : in     FILE_MODE;
                       NAME : in     STRING;
                       FORM : in     STRING    := ""
                      );

     procedure CLOSE  (FILE : in out FILE_TYPE);
     procedure DELETE (FILE : in out FILE_TYPE);
     procedure RESET  (FILE : in out FILE_TYPE; MODE in
     FILE_MODE);
     procedure RESET  (FILE : in out FILE_TYPE);

     function  MODE   (FILE : in FILE_TYPE) return FILE_MODE;
     function  NAME   (FILE : in FILE_TYPE) return STRING;
     function  FORM   (FILE : in FILE_TYPE) return STRING;

     function  IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

--   Control of default input and output files

     procedure SET_INPUT (FILE : in FILE_TYPE);
     procedure SET_OUPUT (FILE : in FILE_TYPE);

     function  STANDARD_INPUT      return FILE_TYPE;
     function  STANDARD_OUTPUT     return FILE_TYPE;

     function  CURRENT_INPUT       return FILE_TYPE;
     function  CURRENT_OUTPUT      return FILE_TYPE;

     --   specification of line and page lengths

     procedure SET_LINE_LENGTH  (FILE : in FILE_TYPE; TO : in COUNT);
     procedure SET_LINE_LENGHT  (TO : in COUNT);

     procedure SET_PAGE_LENGTH  (FILE : in FILE_TYPE; TO : in COUNT);
     procedure SET_PAGE_LENGTH  (                     TO : in COUNT);

     function  LINE_LENGTH      (FILE : in FILE_TYPE) return COUNT;
     function  LINE_LENGTH                            return COUNT;

     function  PAGE_LENGTH      (FILE : in FILE_TYPE) return COUNT;
     function  PAGE_LENGTH                            return COUNT;

     --   Column, Line, and Page Control
```

```
          procedure NEW_LINE    (FILE  :  in  FILE_TYPE;  SPACING  :  in
          POSITIVE_COUNT :=
1);
          procedure NEW_LINE    (                     SPACING    :    in
          POSITIVE_COUNT :=
1);
          procedure SKIP_LINE   (FILE  :  in  FILE_TYPE;  SPACING  :  in
          POSITIVE_COUNT :=
1);
          procedure SKIP_LINE   (                     SPACING    :    in
          POSITIVE_COUNT :=

          function  END_OF_LINE (FILE : in FILE_TYPE) return BOOLEAN;
          function  END_OF_LINE                       return BOOLEAN;

          procedure NEW_PAGE    (FILE : in FILE_TYPE);
          procedure NEW_PAGE                        ;

          procedure SKIP_PAGE   (FILE : in FILE_TYPE);
          procedure SKIP_PAGE                       ;

          function  END_OF_PAGE (FILE : in FILE_TYPE) return BOOLEAN;
          function  END_OF_PAGE                       return BOOLEAN;

          function  END_OF_FILE (FILE : in FILE_TYPE) return BOOLEAN;
          function  END_OF_FILE                       return BOOLEAN;

          procedure SET_COL     (FILE : in FILE_TYPE; TO : in POSITIVE_COUNT);
          procedure SET_COL                         ; TO : in POSITIVE_COUNT);

          procedure SET_LINE    (FILE : in FILE_TYPE); TO : in POSITIVE_COUNT);
          procedure SET_LINE                         ; TO : in POSITIVE_COUNT);

          function  COL         (FILE : in FILE_TYPE; return POSITIVE_COUNT);
          function  COL                             ; return POSITIVE_COUNT);

          function  LINE        (FILE : in FILE_TYPE; return POSITIVE_COUNT);
          function  LINE                            ; return POSITIVE_COUNT);

          function  PAGE        (FILE : in FILE_TYPE; return POSITIVE_COUNT);
          function  PAGE                            ; return POSITIVE_COUNT);

          -- Character IO

          procedure GET  (                     ITEM : out CHARACTER);
          procedure GET  (FILE : in FILE_TYPE;  ITEM : out CHARACTER);
          procedure PUT  (                     ITEM :  in CHARACTER);
          procedure PUT  (FILE : in FILE_TYPE;  ITEM :  in CHARACTER);

          --   STRING IO

          procedure GET  (                     ITEM : out STRING);
          procedure GET  (FILE : in FILE_TYPE;  ITEM : out STRING);
          procedure PUT  (                     ITEM :  in STRING);
          procedure PUT  (FILE : in FILE_TYPE;  ITEM :  in STRING);
```

F.8.3.1 (Continued)

```
          procedure GET_LINE  (FILE : in FILE_TYPE; ITEM : out STRING;
          LAST : out NATURAL);
          procedure GET_LINE  (                     ITEM : out STRING;
          LAST : out NATURAL);

          procedure PUT_LINE  (FILE : in FILE_TYPE; ITEM :  in STRING);
          procedure PUT_LINE  (                     ITEM :  in STRING);

          --   Generic Package for IO of Integer Types

     generic
          type NUM is range ,.;
     package INTEGER_IO is

          DEFAULT_WIDTH   :  FIELD        := NUM°WIDTH;
          DEFAULE_BASE    :  NUMBER_BASE :=        10;

          procedure GET  (FILE : in FILE_TYPE; ITEM: out NUM; WIDTH : in FIELD
          := 0);
          procedure GET  (                     ITEM: out NUM; WIDTH : in FIELD
          := 0);

          procedure PUT  (FILE      : in FILE_TYPE;
                          ITEM      : in NUM;
                          WIDTH     : in FIELD := DEFAULT_WIDTH
                          BASE      : in NUMBER_BASE := DEFAULT_BASE);

          procedure GET   (FROM     :  in  STRING;  ITEM  :  out  NUM:  LAST  :  out
                           POSITIVE);
          procedure PUT   TO        : out STRING;
                          ITEM      :   in NUM;
                          WIDTH     :   in NUMBER_BASE := DEFAULT_BASE);

     end INTEGER_IO;

     --   Generic Package for Input-Output of Enumeration Types

generic
     type ENUM is (,.);
  package ENUMERATION_IO is

     DEFAULT_WIDTH         : FIELD    := 0;
     DFEAULT_SETTING       : TYPE_SET := UPPER_CASE;

     procedure GET  (FILE : in FILE_TYPE;    ITEM : out ENUM);
     procedure GET  (                        ITEM : out ENUM);

     procudure PUT  (FILE      : in FILE_TYPE;
                     ITEM      : in ENUM;
                     WIDTH     : in FIELD     := DEFAULT_WIDTH
                     SET       : in TYPE_SET  := DEFAULT_SETTING);

     procedure PUT  (ITEM      : in ENUM;
                     WIDTH     : in FIELD     := DEFAULT_WIDTH
                     SET       : in TYPE_SET  := DEFAULT_SETTING);
     procedure GET  (FROM      :  in STRING;  ITME : out ENUM;
                     LAST      : out POSITIVE);
     procudure PUT   TO        : out STRING;
                     ITEM      :  in ENUM;
                     SET       :  in TYPE_SET := DEFAULT_SETTING);

end ENUMERATION_IO;

generic
     type NUM is digits ,.;
package FLOAT_IO is

     DEFAULT_FORE    :   FIELD  :=               2;
     DEFAULT_AFT     :   FIELD  := NUM°DIGITS - 1;
     DEFAULT_EXP     :   FIELD  :=               3;

     procedure GET         (FILE : in FILE_TYPE;
                             ITEM : out NUM;
                            WIDTH:  in FIELD:= 0);

     procedure GET         (FILE : in FILE_TYPE;
                             ITEM : out NUM;
                            WIDTH:  in FIELD:= 0);

     procedure PUT         (FILE :  in FILE_TYPE;
                             ITEM :  in NUM;
                            FORE :  in FIELD := DEFAULT_FORE;
                            AFT  :  in FIELD := DEFAULT_AFT;
                            EXP  :  in FIELD := DEFAULT_EXP);
     procedure PUT         (ITEM :  in NUM;
                            FORE :  in FIELD := DEFAULT_FORE;
                            AFT  :  in FIELD := DEFAULT_AFT;
                            EXP  :  in FIELD := DEFAULT_EXP);

     procedure GET         (FROM :  in STRING; ITEM : out NUM;
                            LAST : out POSITIVE);
     procedure PUT         (TO   : out STRING;
                             ITEM :  in NUM;
                            FORE :  in FIELD := DEFAULT_AFT;
                            EXP  :  in FIELD := DEFAULT_EXP);

end FLOAT_IO;

generic
     type NUM is delta ,.;
package FIXED_IO is

     DEFAULT_FORE    :    FIELD := NUM°FORE;
     DEFAULT_AFT     :    FIELD := NUM°AFT;
     DEFAULT_EXP     :    FIELD := 0;

     procedure GET         (FILE : in FILE_TYPE;
                             ITEM : out NUM;
                            WIDTH:  in FIELD:= 0);
```

F.8.3.1 (Continued)

```
     procedure GET       (
                    ITEM : out NUM;
                         WIDTH:  in FIELD:= 0);

     procedure PUT       (FILE :  in FILE_TYPE;
                          ITEM :  in NUM;
                          FORE :  in FIELD := DEFAULT_FORE;
                          AFT  :  in FIELD := DEFAULT_AFT;
                          EXP  :  in FIELD := DEFAULT_EXP);

     procedure PUT        ITEM :  in NUM;
                          FORE :  in FIELD := DEFAULT_FORE;
                          AFT  :  in FIELD := DEFAULT_AFT;
                          EXP  :  in FIELD := DEFAULT_EXP);

     procedure GET       (FROM :  in STRING; ITEM : out NUM;
                          LAST : out POSITIVE);
     procedure PUT       (TO   : out STRING;
                          ITEM :  in NUM;
                          FORE :  in FIELD := DEFAULT_AFT;
                          EXP  :  in FIELD := DEFAULT_EXP);

end FIXED_IO;

STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;
LAYOUT_ERROR : exception renamer IO_EXCEPTIONS.LAYOUT_ERROR;

private
     type FILE_TYPE is new BASIC_IO_TYPES.FILES_TYPE;

end TEXT_IO;
```

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..125 =>'A', 126 =>'1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..125 =>'A', 126 =>'2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..63 | 65..126 =>'A', 64 =>'3')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..63 | 65..126 =>'A', 64 =>'4')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..123 =>'0', 124..126 =>'298')

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..120 =>'0', 121..126 =>'69.0E1')
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1..63 =>'A')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1..62 =>'A', 63 =>'1')
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..106 =>' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	35
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	X}]!@#$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100000.0

Name and Meaning	Value
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	ILLEGAL!@#$%^
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	ILLEGAL&()_+=
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32_768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-200000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	18
$MAX_IN_LEN Maximum input line length permitted by the implementation.	126
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648

TEST PARAMETERS

Name and Meaning	Value
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1=>'2',2..122=>'0',123..125=>"11:")
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3=>"16:",4..122=>'0',123..126=>"F.E:")
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1=>'"',2..125 =>'A',126=>'"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2_147_483_648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFF#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

1. B28003A: A basic declaration (line 36) incorrectly follows a later declaration.

2. E28005C: This test requires that "PRAGMA LIST (ON);" not appear in a listing that has been suspended by a previous "PRAGMA LIST (OFF);"; The Ada Standard is not clear on this point, and the matter will be reviewed by the AJPO.

3. C34004A: The expression in line 168 yields a value outside the range of the target type T, but there is no handler for CONSTRAINT_ERROR.

4. C35502P: The equality operators in lines 62 and 69 should be inequality operators.

5. A35902C: The assignment in line 17 of the nominal upper bound of a fixed-point type to an object raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

6. C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

7. C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may, in fact, raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

8. C35A03E and C35A03R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard does not support this assumption.

9. C37213H: The subtype declaration of SCONS in line 100 is incorrectly expected to raise an exception when elaborated.

10. C37213J: The aggregate in line 451 incorrectly raises CONSTRAINT_ERROR.

11. C37215C, C37215E, C37215G, and C37215H: Various discriminant constraints are incorrectly expected to be incompatible with type CONS.

12. C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

13. C41402A: The attribute 'STORAGE_SIZE is incorrectly applied to an object of an access type.

14. C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

15. C45614C: The function call of IDENT_INT in line 15 uses an argument of the wrong type.

16. A74106C, C85018B, C87B04B, and CC1311B: A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur at lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively.

17. BC3105A: Lines 159 through 168 expect error messages, but these lines are correct Ada.

18. AD1A01A: The declaration of subtype SINT3 raises CONSTRAINT_ERROR for implementations which select INT'SIZE to be 16 or greater.

19. CE2401H: The record aggregates in lines 105 and 117 contain the wrong values.

20. CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.